임베디드SW 기말 프로젝트

임베디드시스템공학과 3학년 202201673 이진성

Embedded SW Final Project

공 통

- 5월 13일 시작이고, 마감은 6월 2(일)일입니다.
- 6월 2일 (일) 오후 4시까지 상세보고서/회로도/모든 라인에 주석문이 들어간 소스 및 프로젝트 폴더 전체/기판사진 여러장/동작상세동영상(화면반은 컴퓨터화면, 화면반은 동작보드가 보이게 편집해서)/프로젝트발표동영상(얼굴과 작품이 같이 나오게)/소스 흐름도를 이러닝에 제출합니다. 동영상 크기가 클 경우 별도 메일로 제출해도 괜찮습니다.
- * 중요 * 모든 함수는 로레벨로 직접 만들어서 사용합니다. HAL 함수와 제공받은 함수 어떤것도 안됩니다. 모두 일일이 지정해서 사용하세요.
  hwinit() 이외에는 어떤 매크로와 변수, 함수도 제공받은 것을 사용하면 안됩니다. 필요한 것은 main 파일 내에 재 정의하고 설계해서 작성하세요.
- 14주차인 6월 3일 (월) 수업시간에 상호 인기 평가합니다. 프로젝트 기간이 짧으니 집중해서 잘 하세요. (일정이 늘어지면 16주차까지 할수도 있는데, 그것보다 3일 마감이 나을겁니다.)
  동작점검을 위해 최종 완성작품은 (일부 또는 전부에 대해) 3일 평가후에 메인보드가 연결된 상태대로 그대로 제출받을 수도 있습니다. 점검후 돌려줍니다.
  6월 4일 실습 수업은 3일 평가에 따라 방법이 달라집니다. 제출된 보드라 하더라도 기말고사를 위해 6월 8일 전에는 다시 돌려줍니다.
- 학과에서 나눠준 보드와 재료들은 16주 금요일(6월 21일)까지 학과 사무실에 반납해야 합니다.
- 포트폴리오로 남기기 위한 기록과 정리를 16주 기간동안 충분히 잘 마무리하고, 모든것을 분리해서 정리한 뒤에 반납하기 바랍니다.

평 가

- 각 등급별로 상대 평가와 절대 평가를 같이 진행합니다. (기준이상이 되어야 하고, 그 들끼지 상대평가합니다.)
- 모든 사람들의 아이디어는 공유해서는 안됩니다. 같은 프로젝트하는 사람들끼리 공유된 내용이 있거나 공유된 방법이 있으면 모두 큰 감점 받습니다.
- 우연히 겹치는 아이디어가 있는 것과, 아이디어가 공유된것은 완전히 다릅니다. 그러니 처음부터 그 어떤것도 공유하지 말고 스스로 진행합니다.
- 공유했다는 판단 유무는 결과와 내용을 보고 교수가 평가합니다. 결과적으로 오판이 있을수도 있을지도 모르겠으나, 그만큼 유사하기는 어렵습니다. 그러니 처음부터 개별로 진행하세요.
- 같이 논의하고 준비하다보면 결국 비슷해 집니다. 평가할 때 제일 먼저 진행할 것이 방법과 내용의 유사도입니다. 먼저 걸러내고 시작합니다. 주의하기 바랍니다.

내 용

1번(S-1번)
- 5단계의 경사를 타고 내려오는 X 간격의 N 개의 구슬을 코딩에 정한 3개의 홀에 차례대로 떨어뜨리기.
- 경사는 5단계를 미리 정하면 되고, Hole 은 3개로 지정
- X, N 은 몇가지를 선택해도 되고, 센싱해서 계산해도 됨.
- 3개의 홀에 떨어뜨리는 것은 코드에 정해서 바꿔가며서 동작시키면 됨.
- 홀에 떨어뜨린 공의 개수는 카운트 해서 숫자로 표시하면 됨. 카운트 방법은 다양한 아이디어
- 이 그룹은 마지막에 면담이 있을 수 있음
- 발광다이어드, 포토트랜지스터 등을 이용해서 공학적으로 속도 측정하여 처리하고 완성하면 플러스 알파
- 공학적인 동작과 SW적인 구성이 잘 갖춰져야 함.
- 임베디드 SW 이므로, 매우 정밀하고 정확하게 완성되어야 함.
- 가장 엄격한 기준으로 점검함

목 차

Ⅰ. 프레임 설계

1.1. 경사로 기초 뼈대

경사로를 설계하는데에 기초 뼈대로 사용하기 위해 ~~연구실에 방치되어 있던~~ 해체된 책상의 프레임을 사용하기로 하였다. 높이는 약 1.2m이다.

그림 1-1-1. 5월 14일, 경사로 제작에 사용되는 책상 프레임

1.2. 경사로 그리드 설계

경사로를 설계하는데에 프레임만 있는 상태로는 설치할 경사를 지지할 수 없는 상황이다. 때문에 프레임의 양쪽에 걸치는 형태의 그리드를 설계해야 한다.

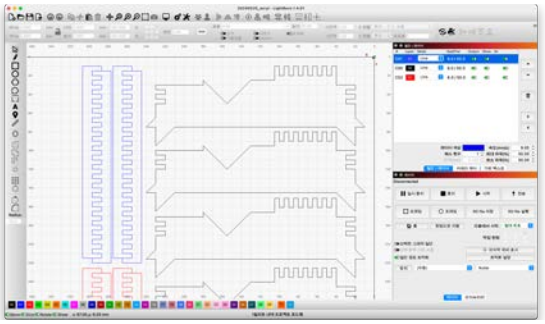


그림 1-2-1. 5월 20일, 프레임 그리드 설계

그리드는 아크릴을 커팅하여 사용하였으며, 설계를 위해 LightBurn을 사용하였다. 아크릴은 5T 투명 아크릴을 사용하였다. 깊이가 10mm가 되도록 구조물을 만들어 각 파츠를 끼워 조립할 수 있도록 하였다.

그림 1-2-2. 5월 20일, 아크릴 그리드 조립

아크릴 파츠 간 끼워 맞추는 형식의 구조를 통해 각 그리드들이 일정한 간격을 유지하도록 설계하였다.

그림 1-2-3. 5월 20일, 프레임과 그리드 결합

그림 1-2-3은 프레임에 아크릴 그리드를 결합한 모습이다.

1.3. 경사로 설계

경사로 또한 아크릴로 설계하였는데, 2T 투명 아크릴을 사용하였으며, 단순히 폭이 30mm가 되도록 커팅하였다.

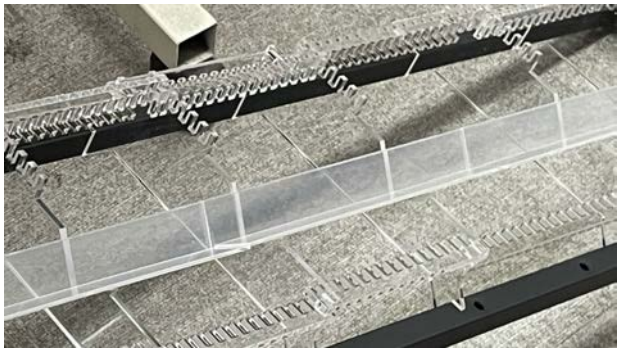


그림 1-3-1. 5월 20일, V자 아크릴 경사로

그리드를 설계할 때 V자로 구조물을 만들었으며, 해당 위치에 경사로가 위치하게 된다. 그림 1-3-1에서는 별도로 고정이 되지 않은 상태로 그리드 위에 안착되어 있는 상황이다. 해당 상황에서는 중력에 의해 약간의 진동 등에 의해서도 움직이거나 떨어질 수 있는 상황이다.

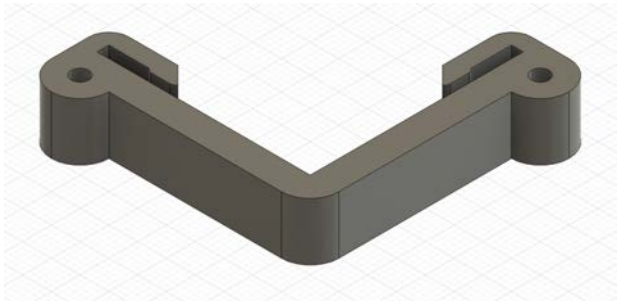


그림 1-3-2. 5월 22일, 경사로 클립

때문에 그림 1-3-2와 같이 경사로와 동일하게 90도의 각도를 갖는 아크릴 클립을 설계하였으며, 3D프린터로 출력하여 사용하였다.

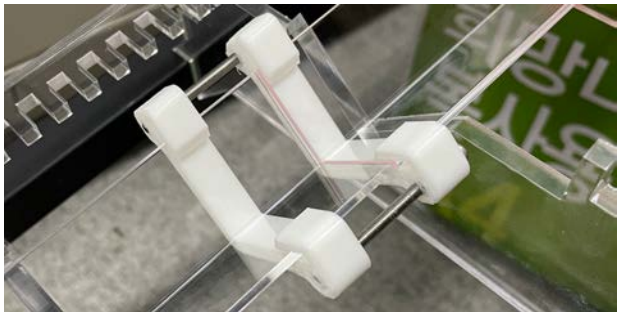


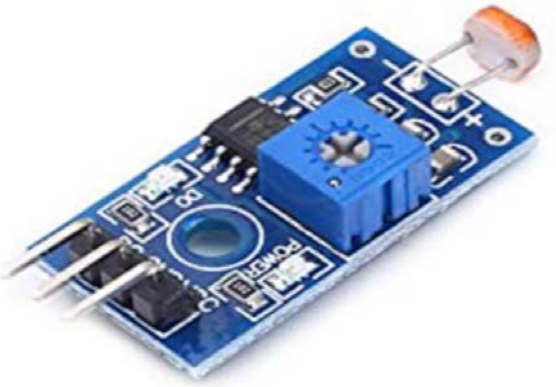
그림 1-3-3. 5월 22일, 클립을 이용한 경사로 결합

아크릴 클립은 각 2개씩의 경사로에 대해 중간부분과 각 끝에 사용되었다. 중간에 사용된 클립은 경사를 지지하는 역할을 하며, 양 끝에 사용된 클립은 다른 경사로의 클립과 M3 나사를 통해 결합되어 여러 경사로들이 안정적으로 연결되도록 하였다.

II. 센서 설계

2.1. 감지 방식 결정

구슬은 경사를 타고 내려오며, 이를 접촉없이 감지하기 위해서는 빛을 사용하는 방법이 가장 일반적이고 안정적일 것이다. 때문에 레이저와 CdS를 사용하고자 한다.



기본적으로 CdS는 광량에 따라 아날로그 신호를 내보내는데, 이는 프로젝트를 진행함에 있어 별도의 로직을 작성해야 하는 부분으로 번거로움이 존재한다. 이 때문에 인터넷을 서치하는 과정에서 별도 회로를 통해 CdS 입력 데이터를 디지털 신호로 변환하도록 설계된 센서를 찾을 수 있었으며, 해당 센서를 사용하여 감지 센서를 설계한다.

레이저는 칩저항이 내장된 5v의 레이저를 사용하였다.

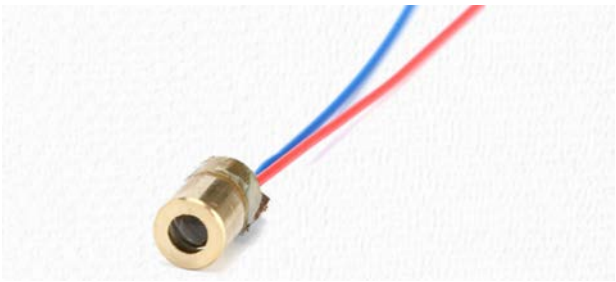


그림 2-1-2. 5v 적색 LED

2.2. 센서 사이즈 체크용 더미 제작

2.1.에서 제시한 두 개의 모듈을 사용하기 위해 각 모듈의 사이즈를 체크한 후 3D프린터 출력물을 사용하였을 때의 공차를 고려하여 정밀하게 조립하기 위한 더미를 설계하였다. 본 프로젝트에서의 3D 설계는 Fusion360을 사용하여 설계되었다.

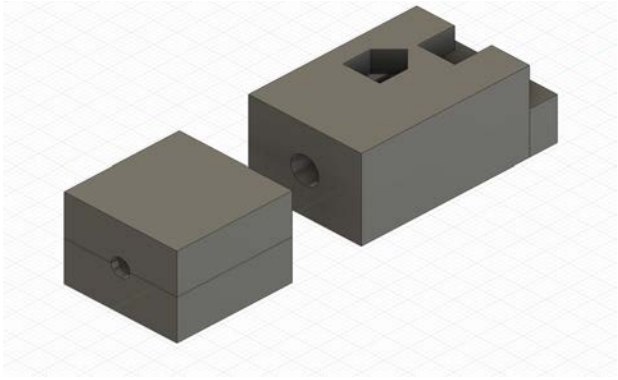


그림 2-2-1. 5월 18일, 더미 설계

레이저는 지름만을 고려하였으며, CdS가 들어가는 위치는 감도 조정을 위한 가변저항을 조립 후에도 조정할 수 있도록 상단에 구멍을 뚫어 설계하였다. 또한 점퍼선을 연결하고 센서가 밖으로 나오지 않도록 후방에 나사 조립형 캡을 동시에 설계하였다. 각 핀의 레이블이 보이게끔 핀헤더 주변의 보이는 영역을 넓게 설계하였다.

2.3. 센서 설계

2.2의 더미를 바탕으로 센서를 설계하였다. 레이저와 센서부는 M3 나사를 사용하는 조립형이다

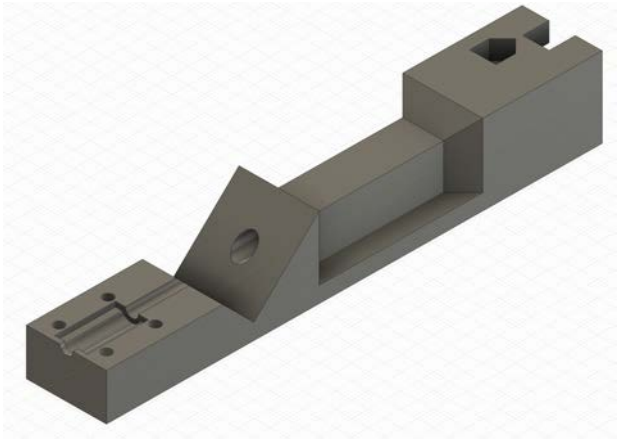


그림 2-3-1. 5월 18일, 센서 구조물 설계

그림 2-3-1의 구조를 출력하고 센서와 레이저를 배치하면 다음과 같다.

그림 2-3-2. 5월 19일, 센서 배치

그림 2-3-2의 레이저 우측을 보면 V자 형태가

보이는데, 이는 아크릴 그리드와 동일한 위치로 설계되어 있으며, 구슬이 경사를 타고 내려올 때 센서가 감지할 수 있도록 하였다.

CdS가 배치되는 곳과 V자 경사의 거리가 어느정도 있는데, 이는 CdS가 외부 빛의 영향을 최소한으로 받게 하기 위해 통로를 길게 설계한 것이다.

2.4. 센서 회로 설계

본 센서에서는 CdS와 레이저 두가지 모듈에 전원을 공급하고 센서값을 읽어와야 하는데, 둘 다 5v의 전원을 지원하므로 하나의 전원을 통해 작동시킬 수 있도록 설계할 경우 회로 측면에서 더욱 효율적인 설계가 가능하다. 때문에 5개의 핀을 3개로 줄여 회로 설계 및 와이어링을 용이하게 하였다.

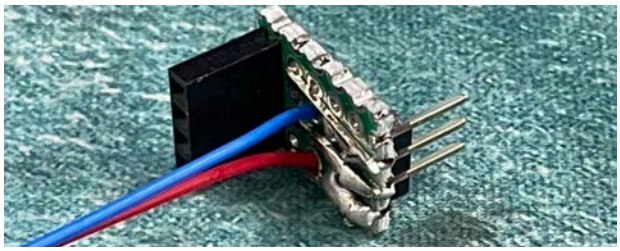


그림 2-4-1. 5월 19일, 센서 회로

그림 2-4-1은 레이저와 CdS의 전원공급을 동시에 하기 위해 만들어진 회로인데, female pin은 점퍼선을 통해 CdS와 연결되며, male pin은 전원과 GPIO에 연결된다. 이미지 상에 보이는 와이어는 레이저 자체에 연결되어 있는 와이어이다. 세 줄의 와이어 중 가운데의 와이어가 데이터 핀이며, 양쪽에 해당하는 와이어가 전원선이다.

2.5. 센서 작동 확인

회로에 CdS와 레이저 모듈을 조립한 후 전원을 연결하면 레이저의 붉은 빛과 CdS 자체에 연결된 초록색의 전원 LED와 digital value를 표시하는 LED가 켜지는 것을 볼 수 있다.

그림 2-5-1. 5월 19일, 센서 값이 1인 경우

그림 2-5-2. 5월 19일, 센서 값이 0인 경우

그림 2-5-1과 2-5-2는 센서에 전원을 연결한 상태에서 구슬이 지나가는 위치를 손가락으로 막지 않은 상태와 막은 상태이다. 그림 2-5-1에서는 레이저 빛이 CdS에 도달하는 것을 볼 수 있으며, CdS의 digital LED가 켜진 것을 볼 수 있다. 반대로 그림 2-5-2에서는 레이저 빛이 CdS에 도달하지 못해 digital LED가 꺼진 것을 볼 수 있다.

2.6. 센서 조립

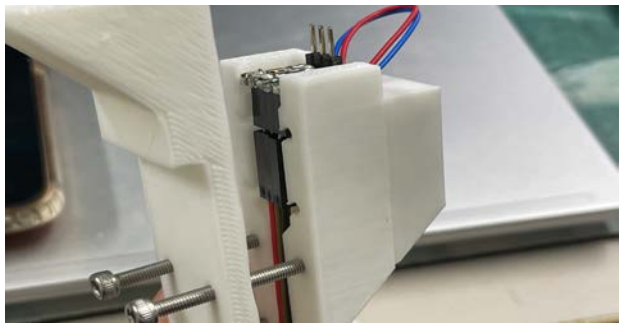


그림 2-6-1. 5월 20일, 센서 회로부 결합 방식

앞의 설계와 와이어링을 종합하여 조립하는 과정이 필요하다. 이를 위해 그림 2-3-1의 설계에서 점퍼선이 지나갈 수 있는 홈을 만들고 회로가 들어갈 수 있는 위치를 설계하였다.

또한 그림 2-3-1에서는 CdS와 레이저 모듈을 결합할 수 있는 구조물만으로 구성되어 있는데, 이를 경사로 프레임에 결합할 수 있도록 추가 설계가 필요하다. 이를 위해 그림 1-2-1과 동일한 규격이 되도록 V자 홈의 위치에 맞춰 프레임에 걸칠 수 있도록 설계하였다. 이는 3D프린터의 출력 가능한 크기와 유지보수 용이성을 위해 추가 조립이 필요한 형태로 제작하였으며, M3 나사를 사용하여 조립할 수 있도록 설계하였다.

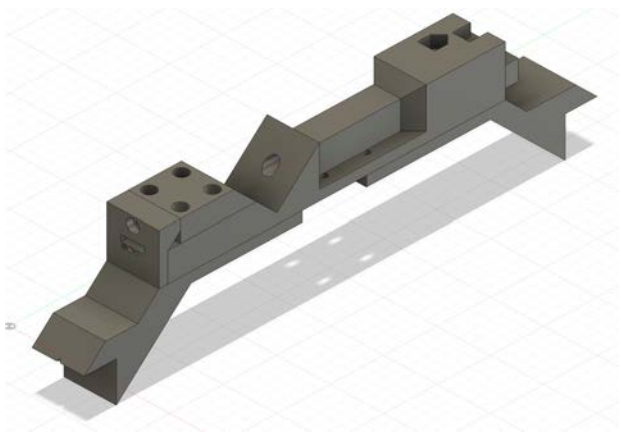


그림 2-6-2. 5월 21일, 센서 구조물 설계 최종본

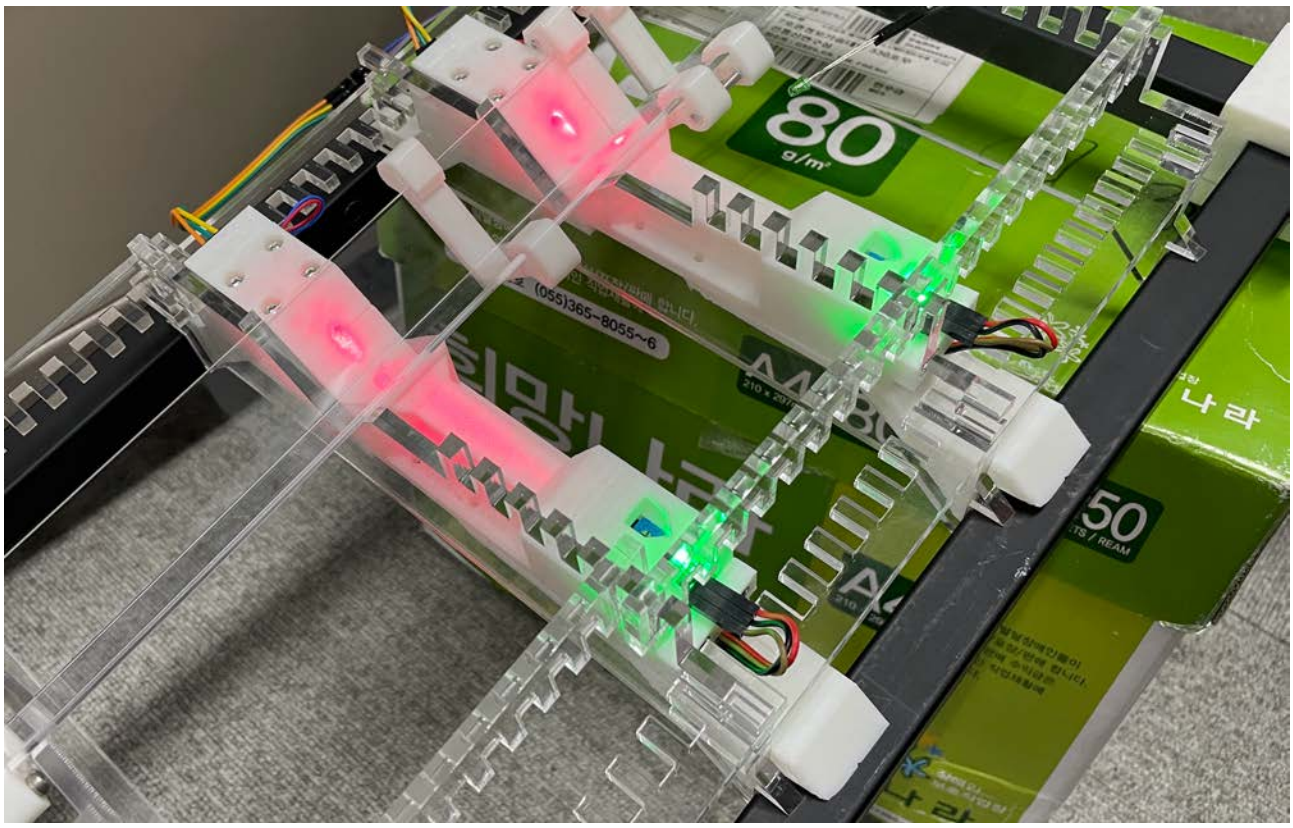


그림 2-6-3. 5월 22일, 센서 설치

2-6-2의 설계를 모두 출력하여 조립한 후 프레임에 부착, 전원을 연결하면 그림 2-6-3과 같은 형상을 띈다.

센서는 아크릴 그리드의 아래에 배치되는 것을 볼 수 있으며, CdS 방향에는 회로와 연결하기 위한 점퍼선을 볼 수 있다. 또한 경사로에 투명아크릴을 사용하여 레이저 빛이 경사로가 막고 있더라도 안정적으로 CdS에 도달하는 것을 볼 수 있다.

센서 설계에는 총 6개의 3D 프린터 출력물이 사용되었다.

III. 액추에이터 설계

3.1. 액추에이터 부품 선정

액추에이터로 서보모터를 사용하려고 하였으며, 이를 통해 일반적으로 아두이노에서 많이 사용하는 SG90과 MG90S 90도 제품을 찾을 수 있었다. 설계할 액추에이터가 SG90으로 구동할 수 있을지에 대한 의문이 들어 SG90과 MG90S 90도 모델을 모두 구입하였으며, SG90을 테스트 하는 과정에서 각도의 불안정성을 인지하고 MG90S 90도 모델을 사용하기로 결정하였다.

그림 3-1-1. MG90S 서보모터

MG90S 서보모터는 20ms 주기의 50Hz PWM 신호로 제어가 가능하며, 데이터 시트 상에서는 1.5ms에서 2ms 펄스에서 0도에서 90도의 각도로 제어가 가능하다고 기재되어있지만 실제로는 1ms에서 2ms 펄스에서 0도부터 90도의 각도로 제어가 가능함을 확인하였다.

3.2. 액추에이터 구동 방식

액추에이터의 역할은 원하는 시점에 구슬을 기존 경사에 그대로 흘려보내거나 홀에 빠뜨리는 것이다.

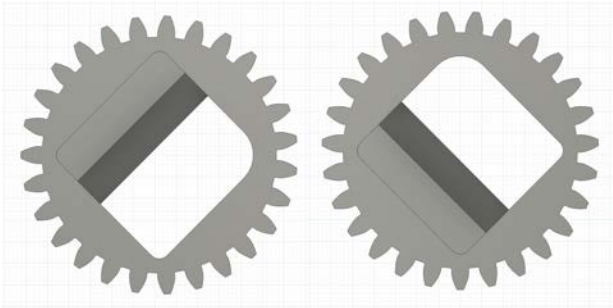


그림 3-2-1. 5월 22일, 회전장치 설계

이를 구현하기 위해 원기둥 형태의 구조물을 설계하였다. 그림 3-2-1은 해당 구조물의 단면에 해당하는데 90도 회전하였을 때를 동시에 보여주고 있다. 경사로와 동일한 90도 각도의 구조를 가지고 있으며, 좌측의 경우 경사로를 따라 그대로 내려가도록 설계된 것을 알 수 있다.

반대로 우측의 경우에는 구조물에 의해 구슬의 경로가 차단되는 모습을 보이는데, 차단 후 측면으로 빠지도록 설계한 것이다. 그림 3-2-2는 회전장치에서 구슬의 경로가 차단되었을 때의 메커니즘을 보여주고 있는데, 차단 경로의 측면을 통해 옆으로 빠질 수 있도록 설계됨을 알 수 있다. 또한 45도의 경사가 있어 구슬이 차단됨과 동시에 빠르게 튕겨나갈 수 있도록 설계하였다.

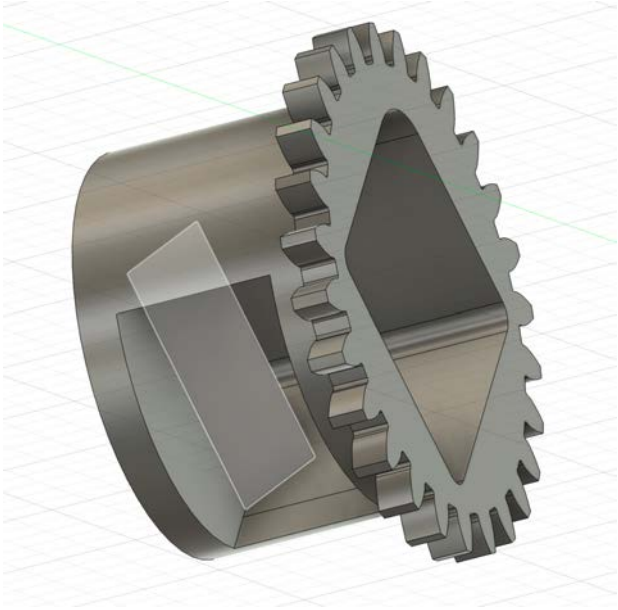


그림 3-2-2. 5월 22일, 회전장치 하단 시점

회전장치는 동일한 사이즈의 기어와 맞물려 회전하도록 설계되었는데, 이 기어는 Fusion360의 기본 Add-in인 Spur Gear를 사용하여 만들어졌다.

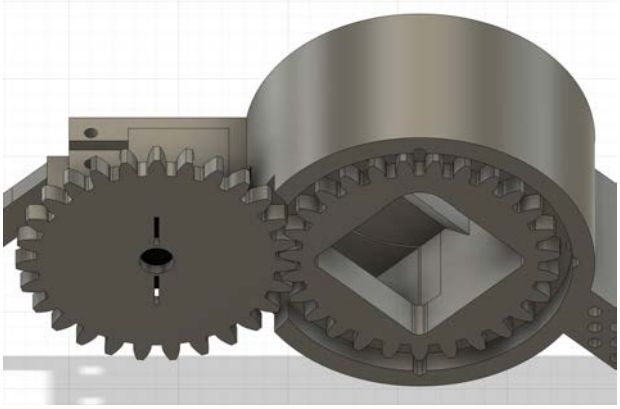


그림 3-2-3. 5월 23일, 액추에이터 구동부

그림 3-2-1과 3-2-2에서 보인 구조물은 원기둥 형태의 구조물 내에서 회전하며, 측면의 동일한 사이즈의 기어가 MG90S 서보모터와 결합되어 회전하게되면 동일한 회전비로 회전하도록 설계하였다.

3.3. 액추에이터 설계

3.2.에서 액추에이터 설계의 일부를 구동 방식과 함께 기술하였다. 그림 3-2-3에서 대부분의 구조를 보였다. 그림 3-2-n에서 보인 기어 내의 사각형 구조물은 그림 1-2-1의 설계에서 보인 그리드의 V자의 위치와 동일하게 설계하였기 때문에 2T 아크릴을 사용함을 감안하여 특히 통과하여 지나가는 3-2-1의 왼쪽에 해당하는 상황에서는 아크릴 두께인 2mm만큼의 두께가 더 필요하며, 이를 위해 추가적인 파츠를 제작하여 부착하였다.

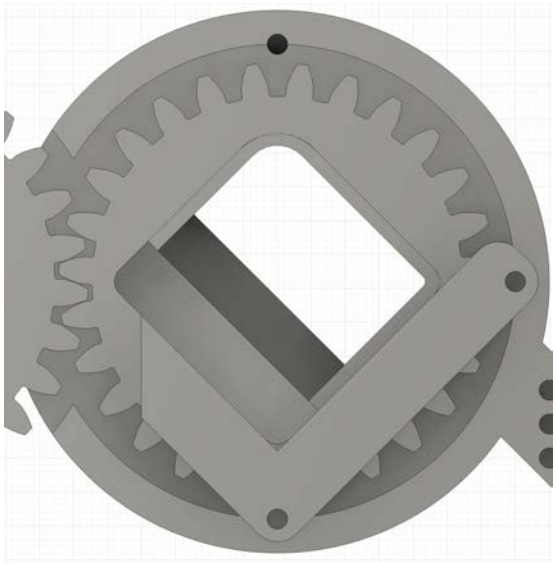


그림 3-3-1. 5월 26일, 액추에이터 2mm 파츠와 경사 받침

또한 기어의 이탈을 방지하고, 경사로의 처짐을 방지하고 액추에이터와 동일한 높이를 유지하기 위해 받침을 추가하여 M3 나사로 고정하게 하였다.

액추에이터의 위치를 맞추기 위해 그림 2-6-2의 프레임 거치 위치와 동일한 구조를 설계하였다.

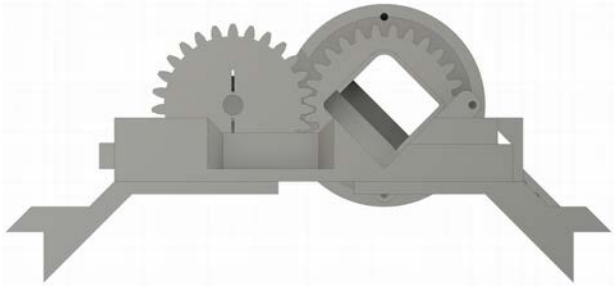


그림 3-3-2. 5월 26일, 액추에이터와 센서 설계 오버랩

그림 3-2-1의 구조를 통해 홀에 빠뜨리는 경우는 액추에이터 설계부에도 회전장치와 동일한 위치에 구멍을 통해 구슬이 빠져나갈 수 있도록 설계하였다.

그림 3-3-3. 5월 26일, 액추에이터와 하단부

액추에이터 설계에도 총 6개의 3D 프린터 출력물이 사용되었다.

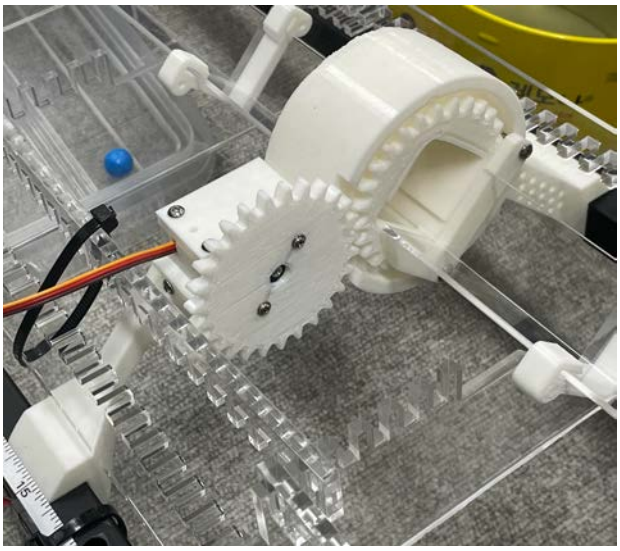


그림 3-3-4. 5월 26일, 액추에이터와 설치

Ⅳ. 센서 및 액추에이터 배치 및 배선

4.1. 센서 및 액추에이터 배치 간격

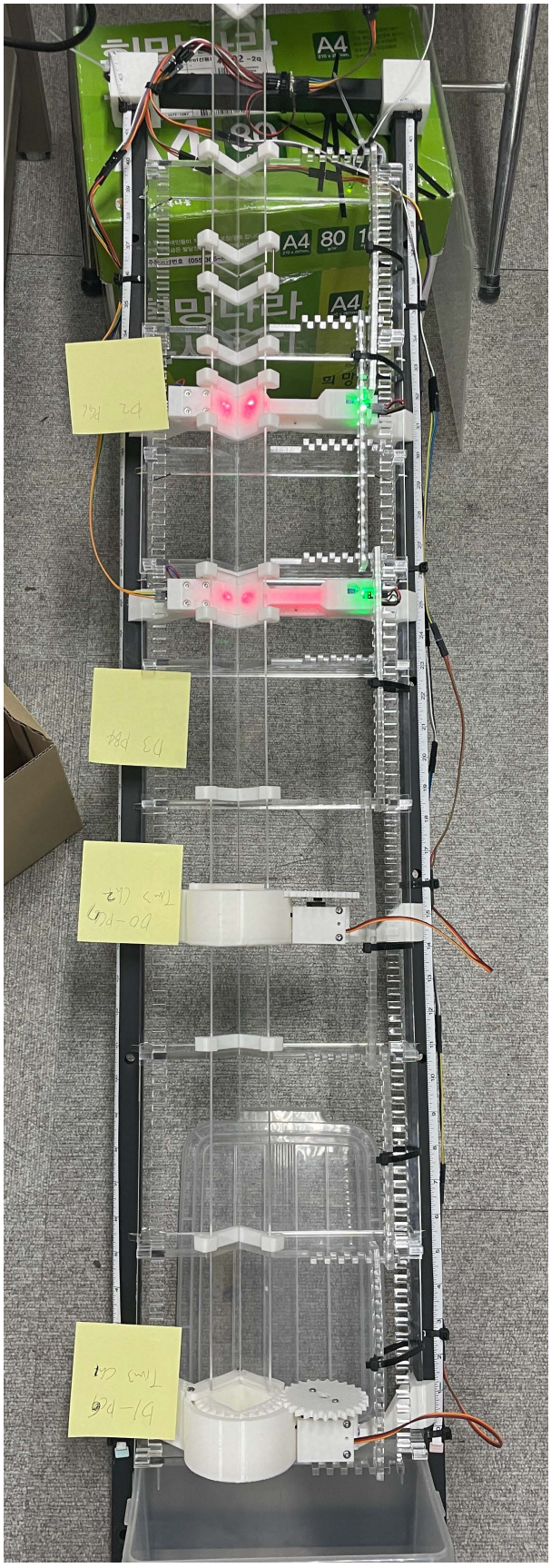


그림 4-1-1. 5월 27일, 센서 및 액추에이터 배치

센서는 레이저가 지나가는 센서의 중앙을 기준으로, 액추에이터는 그림 3-2-2의 45도 경사가 시작하는 액추에이터의 중앙부를 기준으로 거리를 측정하였다.

구슬의 초기 속도는 0이고, 등가속운동을 하기 때문에 시간에 따른 속도는 원점을 지나는 일차함수 꼴을 나타낸다. 이때 t_0의 시간동안 이동한 거리를 S라고 하면 $2t_0$, $3t_0$, $4t_0$일 때 $t=0$일때부터 이동한 거리는 $4S$, $9S$, $16S$가 된다. 동일한 말로 $t=0$일때부터 t_0마다 이동한 거리는 S, $3S$, $5S$, $7S$가 된다. 반대로 말하면 S, $3S$, $5S$, $7S$만큼의 거리를 이동하는데 걸리는 시간은 t_0로 일정하다.

연산의 편의를 위해 이 시간과 거리의 관계를 사용하여 첫 번째 구슬의 출발지점과 두 개의 센서, 두 개의 액추에이터의 간격을 S, $3S$, $5S$, $7S$로 지정하였다. 프레임 내에서 일정한 간격을 유지하도록 하기 위해 해당 간격에 비례하도록 설정하게끔 하였다. S는 2인치에 해당한다. 해당 간격을 맞추기 위해 프레임의 양쪽에 줄자를 고정하였다.

출발지점에서 가까운 첫 번째 센서는 ARD-D2번 핀에 연결되어 있으며, ARD-D2는 PG6핀을 사용한다. 두 번째 센서는 ARD-D3번 핀에 연결되어 있으며, PB4핀을 사용한다. 첫 번째 액추에이터는 ARD-D1에 연결되어 있으며, PC6핀을 사용한다. TIM3의 CH1을 사용한다. 두 번째 액추에이터는 ARD-D0에 연결되어있다. PC7핀에 연결되어 있으며, TIM3의 CH2를 사용하도록 하였다. 디지털 0~3번 핀에 연속적으로 핀을 배치하였다.

표 4-1-1. 기기간 연결 관계

Device	Pin	GPIO	Timer/Interrupt	
Actuator1	ARD-D1	PC6	Timer3	CH1
Actuator2	ARD-D0	PC7	Timer3	CH2
Sensor1	ARD-D2	PG6	EXTI6	
Sensor2	ARD-D3	PB4	EXTI4	

4.2. 공통 핀 수 최소화

센서와 액추에이터는 VCC와 GND, 데이터 핀까지 총 3개의 핀이 공통적으로 요구된다. 2개의 센서와 2개의 액추에이터가 사용되므로, 12개의 와이어가 사용된다. 또한 4개의 데이터 핀은 ARD0~3의 연속된 핀을 사용한다.

이에 4개씩 공통되는 VCC와 GND 핀을 병렬로 연결하여 12개의 핀을 총 6개의 핀만 사용할 수 있도록 와이어링을 하였다. 이를 프레임 상단에 고

정하였으며, 브레드보드를 통하는 와이어 수를 줄였다.

그림 4-1-2. 5월 27일, 센서 및 액추에이터 배선

6개의 핀을 사용하면 4개의 연속된 데이터 핀과 VCC, GND핀을 사용하기 때문에 보드에 와이어를 연결하기가 용이해진다.

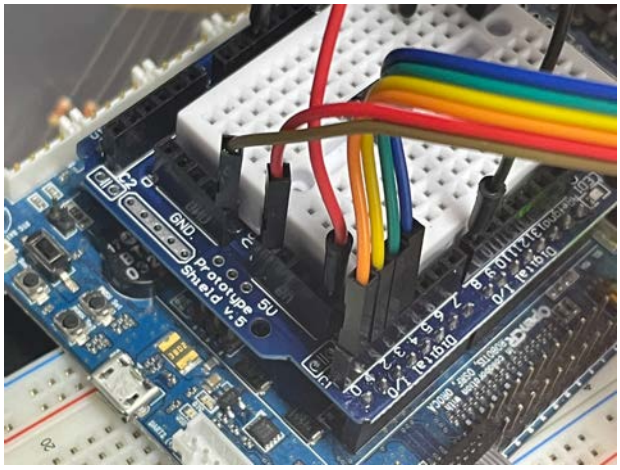


그림 4-1-3. 5월 27일, 보드 와이어링

4.3. 가속도 변화

경사는 대부분이 아크릴을 통해 만들어졌지만, Actuator를 지날때는 3D프린터로 만들어진 경사를 지나게 된다. 때문에 해당 지점에서 가속도가 변화하며, 때문에 $7S$의 거리를 t_0보다 더 긴 시간 동안 주파하게 된다. 때문에 실제로는 Actuator1과 Actuator2 사이의 거리는 $7S$보다 더 짧다. 또한 후술하는 연산에 있어서도 t_0보다 짧은 시간이 걸린다고 가정하고 연산에 적용한다.

Ⅴ. 인터페이스

5.1. 인터페이스 요구 사항에 따른 부품 선정

프로젝트의 요구사항에 따라 N개의 구슬이 순서에 맞춰 홀에 떨어져야 하는데, 이를 코드 상에 구현하는 것은 매 실행때마다 코드를 수정해야하는 문제가 있다. 따라서 외부 입력에 따라서 각 구슬이 떨어지는 홀을 지정하고자 하였다.

이를 위해 4개의 푸쉬 스위치를 사용하는 입력방식을 구상하였으며, 3개의 스위치는 각 홀, 그리고 남은 하나의 스위치는 입력 종료를 의미하도록 구성하였다. 이를 1~3과 4번 스위치라고 한다면

'2 - 3 - 1 - 3 - 4'

의 순서로 스위치를 통해 입력을 받았다면, 총 4개의 구슬이 굴러가고, 첫 번째 구슬부터 두 번째 홀, 세 번째 홀, 첫 번째 홀, 세 번째 홀에 구슬이 차례로 들어가야 한다.

또한 각 홀에 떨어뜨린 구슬의 개수를 각각 나타내야 하는데, 이를 위해 Binary type으로 나타내는 것보다는 7-Segment를 사용하는 것이 더욱 직관적일 것이며, 이를 위해 3개의 7-Segment를 사용하고자 하였다.

7-Segment를 사용함에 있어 IC를 사용하지 않게 될 경우 각 Segment 당 최소 7개의 Digital pin이 필요하기 때문에 IC를 사용하여 Output Digital pin을 줄이고자 하였다. 7447과 74595중 선택하여 사용하려 하였으나 교내에서 여분의 74595를 구할 수 없어 7447을 사용하였다. 세 개의 Segment는 착시를 사용한 방식을 통해 동작하도록 하였다.

5.2. 인터페이스 회로 구성

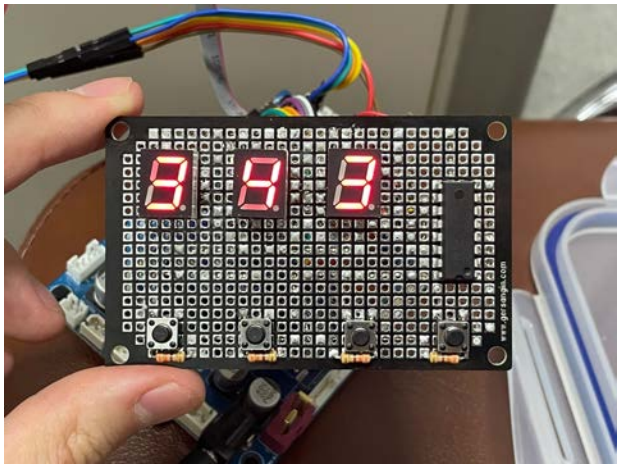


그림 5-2-1. 5월 28일, 인터페이스 레이아웃

회로는 조작의 편의성과 미관 등을 고려하여 브레드보드를 사용하지 않고 설계하였다. 50x80mm인 만능기판을 사용하였다.

가로로 배치한 만능기판에서 하단에 1열로 스위치를, 1~3번 스위치 상단에 7-Segment를, 4번 스위치 상단에 7447 IC를 위치시켰다. 공간 활용 등을 이유로 한 홀에 두 개의 핀이 사용되기도 하였다.

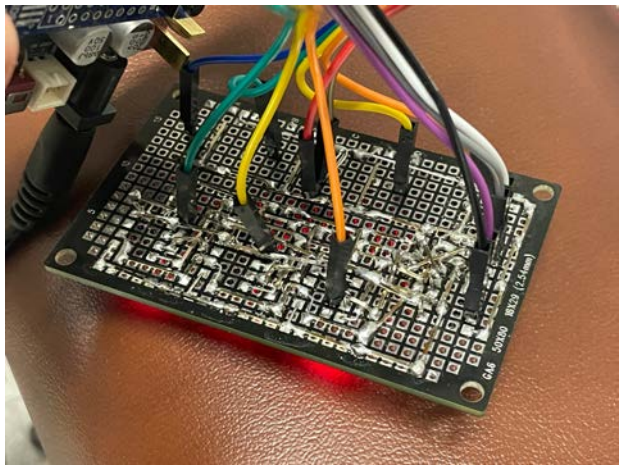


그림 5-2-2. 5월 28일, 인터페이스 후면 와이어링

인터페이스는 총 13개의 와이어가 보드에 와이어링 되는데, 여기에는 VCC와 GND, 스위치 입력 4개, 7447 Input 4개와 7-Segment Selection을 위한 핀 3개가 사용된다.

입력 스위치의 데이터 핀은 기본적으로 330Ω의 저항을 통해 연결되어 0의 값을 가지며, 스위치가 Push되었을 때 VCC와 연결되어 1의 값을 가진다.

7-Segment의 a~g에 해당하는 7개의 입력핀은 3개의 Segment가 모두 하나의 7447의 출력값을 병렬로 연결해 공유하고 있으며, Segment의 전원핀의 값을 통해 각각의 Segment를 켰다 끄는 방식으로 개별 Segment를 제어한다. 이를 위해 3개의 데이터 핀이 각각의 Segment의 전원핀에 연결되어 있다.

IC 7447은 하나만 사용되었는데, A~D에 해당하는 Input Pin은 핀헤더를 연결해 점퍼선을 연결할 수 있도록 하였다. 전원과 LT', BI/RBO', RBI'핀은 공통 전원을 위해 들어오는 VCC, GND 핀에 연결하였다.

5.3. 인터페이스와 보드 연결

13개의 핀을 ARD Digital Pin에 우선적으로 연결하고, 추가적으로 ARD Analog Pin에 연결하였다.

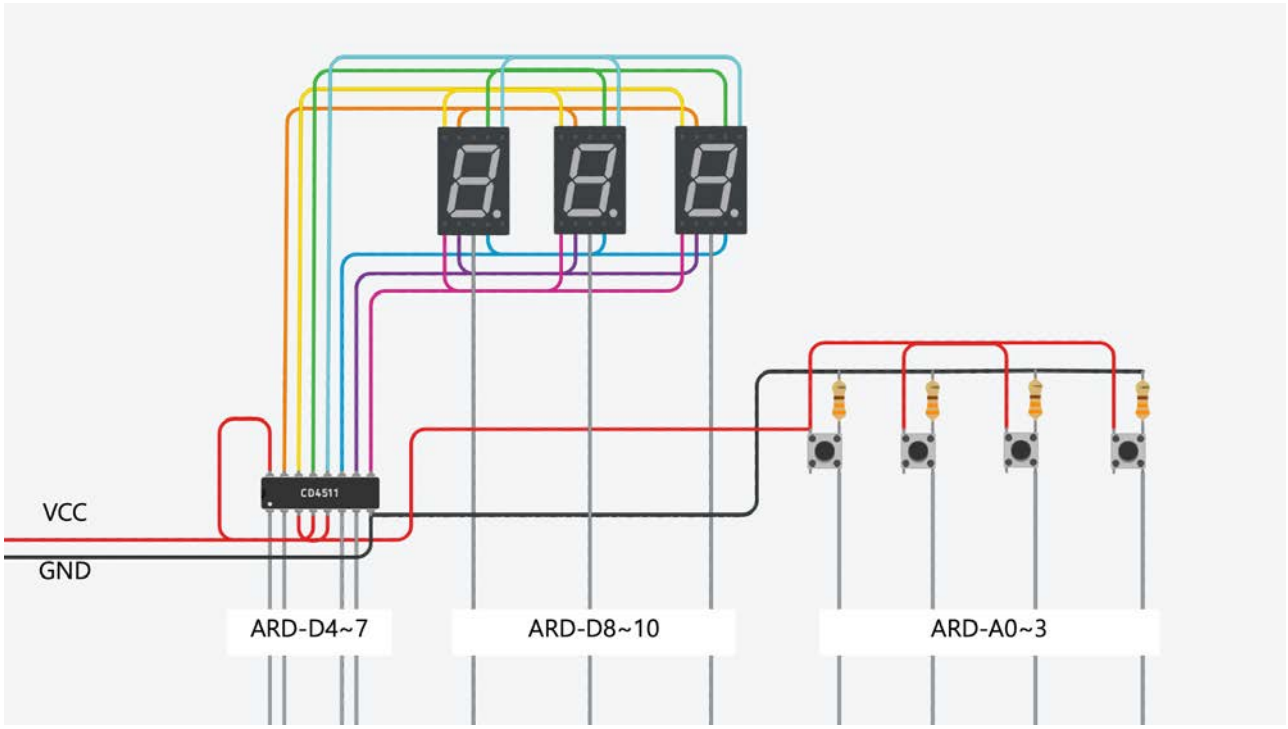


그림 5-3-1. 인터페이스 회로

4.1.에서 ARD-D0~3의 핀을 센서와 액추에이터를 위해 사용하였으며, 해당 핀들은 사용할 수 없다. ~~실습을 진행함에 있어 7447의 입력을 위해 ARD-D4~7을 사용하였어서~~ 동일하게 ARD-D4~7에 7447의 A~D의 Input을 연결하였다.

~~또한 7447 강의 이후 개인적으로 7-Segment 4개를 착시를 이용하여 동작하는 실습을 진행하였으며, 이때 ARD-D8~11을 사용하였다. 이 때문에 동~~일하게 각 Segment의 전원을 제어하는데에 ARD-D8~10을 사용하였다.

표 5-3-1. 인터페이스 보드 연결 관계

Device	Pin	GPIO	I/O
7447-A	ARD-D4	PG7	Output
7447-B	ARD-D5	PA8	Output
7447-C	ARD-D6	PA2	Output
7447-D	ARD-D7	PC1	Output
7-Segment 1	ARD-D10	PB9	Output set on
7-Segment 2	ARD-D9	PA3	Output set on
7-Segment 3	ARD-D8	PC2	Output set on
Switch 1	ARD-A0	PA0	Input push 1
Switch 2	ARD-A1	PF10	Input push 1
Switch 3	ARD-A2	PF9	Input push 1
Switch 4	ARD-A3	PF8	Input push 1

VI. Actuator Code

6.1. Actuator1 Timer Link

TIM3에 연결하는 PC6에 대한 설정을 타이머에 연결하는 Alternate function mode로 설정하고 Ch1에 연결한다.

코드 6-1. 5월 19일, PWM_sample.c

```
void GPIO_Config_TIM3_PC6_ARD1_PWM(){
    *(volatile unsigned int*)(0x40023830) |= 0x01U<<2;
        //RCC_AHB2ENR GPIOC enable
    *(volatile unsigned int*)(0x40020800) |= 0x02U<<(6*2);
        //GPIOC MODER6 Alternate function mode - TIM
    *(volatile unsigned int*)(0x40020820) |= 0x02U<<(6*4);
        //GPIOC_AFRL AFR6 - set AF2 - Link TIM3 Channel2
}
```

6.2. TIM3 Channel 1 Setting

Tim3의 Prescaler와 Auto Reload register를 설정하고 Capture Compare mode 1을 PWM Mode로 설정, PC6로의 출력을 활성화하고 TIM3를 활성화한다.

코드 6-2. 5월 19일, PWM_sample.c

```
void TIM3_PC6_ARD1_PWM_Config(){
    *(volatile unsigned int*)(0x40023840) |= 0x01U<<1;
        //RCC_APB1ENR - TIM3 enable
    *(volatile unsigned int*)(0x40000428) = 108 - 1;
        //set Prescaler value
    *(volatile unsigned int*)(0x4000042C) = 20000 - 1;
        //set Auto Reload register value
        //APB1 block's clock is 108MHz
        // 108,000,000/(108 * 20000) = 50(Hz)
    *(volatile unsigned int*)(0x40000434) = 1000;
        //TIM3 capture/compare register
        // 1000 -> 1ms(ARR = 20000 - 1)
    *(volatile unsigned int*)(0x40000418) |= 0x06U<<4;
        //TIM3_CCMR1 - OC1M
        //Output compare 1 mode
        //PWM mode 1
    *(volatile unsigned int*)(0x40000420) |= 0x01U<<0;
        //TIM3_CCER - CC1E
        //Capture/Compare 1 output enable
    *(volatile unsigned int*)(0x40000400) |= 0x01U<<0;
        //TIM3_CR1 - CEN
        //Counter enable
}
```

6.3. Actuator2 Setting

Actuator2는 PC7핀을 사용하며 동일한 TIM3의 Ch2를 사용한다. 코드의 형식은 6.1.과 6.2.와 동일하며, 핀 설정이 PC6에서 PC7으로 바뀐다. 또한 동일한 TIM3를 사용하며, Channel만 2를 사용하여 두 개의 서보모터에 두 개의 Timer를 사용하는 것이 아닌 하나의 Timer에서 두 개의 Channel을 사용하도록 하였다.

코드 6-3. 5월 19일, PWM_sample.c

```
void GPIO_Config_TIM3_PC7_ARD0_PWM(){
    *(volatile unsigned int*)(0x40023830) |= 0x01U<<2;
    *(volatile unsigned int*)(0x40020800) |= 0x02U<<(7*2);
    *(volatile unsigned int*)(0x40020820) |= 0x02U<<(7*4);
}

void TIM3_PC7_ARD0_PWM_Config(){
    *(volatile unsigned int*)(0x40023840) |= 0x01U<<2;
    *(volatile unsigned int*)(0x40000428) = 108 - 1;
    *(volatile unsigned int*)(0x4000042C) = 20000 - 1;
    *(volatile unsigned int*)(0x40000438) = 1000;
    *(volatile unsigned int*)(0x40000418) |= 0x06U<<12;
        //TIM3_CCMR1 - OC2M
        //Output compare 2 mode
        //PWM mode 1
    *(volatile unsigned int*)(0x40000420) |= 0x01U<<4;
        //TIM3_CCER - CC2E
        //Capture/Compare 2 output enable
    *(volatile unsigned int*)(0x40000400) |= 0x01U<<0;
}
```

6.4. Use servo motor

MG90S 모델은 SG90 모델과 동일하게 50Hz의 PWM 신호를 사용하며, 한 주기는 20ms에 해당한다.(1/50sec) 이때 APB1 블록은 System Clock인 216MHz를 사용하는 것이 아닌 1/2에 해당하는 108MHz를 사용하는 것을 확인하였으며, 이에 따라 Prescaler를 108 - 1, Auto Reload register를 20000 - 1로 설정하였다.

$$108000000/(108*20000) = 50(Hz)$$

이를 통해 Timer3가 50Hz로 동작하게 하였다. 이때 1~2ms에서 0~90도의 동작을 하는 것을 확인하였으며, 이를 위해 두 개의 Capture/Compare register의 값을 1000~2000으로 설정하면 1~2ms로 동작하는 것을 확인할 수 있었다. 이에 따라 아래와 같은 코드로 서보모터를 구동할 수 있다.

코드 6-4. 5월 19일, PWM_sample.c

```
*(volatile unsigned int*)(0x40000434) = 1000;
    //Actuator 1 0 degree
*(volatile unsigned int*)(0x40000434) = 2000;
    //Actuator 1 90 degree
*(volatile unsigned int*)(0x40000438) = 1000;
    //Actuator 2 0 degree
*(volatile unsigned int*)(0x40000438) = 2000;
    //Actuator 2 90 degree
```

Ⅶ. Sensor Code

7.1. Use Interrupt

Polling을 사용하여 입력을 받는 것은 주기적으로 입력값을 체크해야 하고, 주기 내의 입력을 확인할 수 없기 때문에 부적합하다. 이에 따라서 두 개의 센서 입력에는 Interrupt를 사용한다.

7.2. EXTI6 Setting

Interrupt는 센서가 공을 감지하기 시작하는 시점인 rising edge에서 interrupt를 발생시키며, 이를 위해 RTSR의 TR6만 set을 하였다. 또한 Interrupt and Exception table에서 EXTI9_5에 해당하는 값은 23임을 확인하였으며, set 하였다.

코드 7-2. 5월 27일, interrupt.c

```c
void interrupt_setting_PG6_ARD2(){
    *(volatile unsigned int*)(0x40023830) |= 0x00000040U;

        //RCC_GPIOG clock enable
    *(volatile unsigned int*)(0x40021800) &= ~(0x03U<<(6*2));
        //MODER6
    *(volatile unsigned int*)(0x4002180C) &= ~(0x03U<<(6*2));
        //PUPDR

    *(volatile unsigned int*)(0x40023844) |= 0x01U<<14;
        //RCC_APB2ENR - SYSSCFEN
    *(volatile unsigned int*)(0x4001380C) |= 0x06U<<((6%4) * 4);
        //SYSCFG_EXTICR2 - EXTI6 - PG6
    *(volatile unsigned int*)(0x40013C00) |= 0x01U<<6;
        //EXTI_IMR - IM6
    *(volatile unsigned int*)(0x40013C08) |= 0x01U<<6;
        //EXTI_RTSR - TR6   //rising edge trigger

    *(volatile unsigned int*)(0xE000E100) |= 1<<23;
        //NVIC  //Position-23-EXTI9_5
}
```

7.3. EXTI9_5_IRQHandler

EXTI6을 통해 인터럽트가 발생할 경우 EXTI9_5_IRQHandler가 사용된다. PG6에 의해 호출되었는지 PR을 확인, clear, routine을 실행하도록 하였다. service routine은 후술할 것이다.

코드 7-3. 5월 27일, interrupt.c

```c
void EXTI9_5_IRQHandler(){
    if(*(volatile unsigned int*)(0x40013C14) & (0x01U<<6)){
        //PG6
        *(volatile unsigned int*)(0x40013C14) |= 0x01U<<6;
        //EXTI_PR-PR6 clear

        //code for interrupt service
    }
}
```

7.4. EXTI4 Set

Sensor2의 입력을 위한 EXTI4의 세팅은 7.2.와 7.3.의 코드와 유사한 세팅을 사용하며, EXTI4를 사용한다. Interrupt and Exception table에서 EXTI4에 해당하는 값이 10임을 확인하였으며, set 하였다.

코드 7-3. 5월 27일, interrupt.c

```c
void interrupt_setting_PB4_ARD3(){
    *(volatile unsigned int*)(0x40023830) |= 0x01U<<1;
        //RCC_GPIOB clock enable
    *(volatile unsigned int*)(0x40020400) &= ~(0x03U<<(4*2));
    *(volatile unsigned int*)(0x4002040C) &= ~(0x03U<<(4*2));

    *(volatile unsigned int*)(0x40023844) |= 0x01U<<14;
        //RCC_APB2ENR - SYSSCFEN
    *(volatile unsigned int*)(0x4001380C) |= 0x01U<<((4%4) * 4);
        //SYSCFG_EXTICR2 - EXTI4 - PB4
    *(volatile unsigned int*)(0x40013C00) |= 0x01U<<4;
        //EXTI_IMR - IM4
    *(volatile unsigned int*)(0x40013C08) |= 0x01U<<4;
        //EXTI_RTSR - TR4   //rising edge trigger

    *(volatile unsigned int*)(0xE000E100) |= 1<<10;
        //NVIC  //Position-10-EXTI4
}

void EXTI4_IRQHandler(){
    if(*(volatile unsigned int*)(0x40013C14) & (0x01U<<4)){
        //PB4
        *(volatile unsigned int*)(0x40013C14) |= 0x01U<<4;
            //interrupt clear

        //code for interrupt service
    }
}
```

Ⅷ. 비주요 함수

8.1. void setDP, setAP(int pin, int mode)

set*P 함수는 GPIO를 디지털 입출력으로 사용할 때 mode를 세팅하는 함수이다. setDP는 Digital Pin 0~13, setAP는 Analog Pin 0~3에 대한 입출력 setting이 가능하다. Logic 내에서는 mode에 대한 if문을 통해 입력인지 출력인지를 정하는데 이를 위해 OUT과 IN이 1과 0으로 define 되어있다.

mode에 대한 조건문을 통과한 후에는 switch문을 통해 pin의 값에 따른 mode setting logic을 실행한다. 해당 port의 clock을 enable하고, MODER을 General Purpose output 혹은 Input mode로 설정한다. OSPEEDR을 통해 output mode에 대해서는 very high speed로 설정하고, PUPDR을 통해 pull-up으로 설정한다. output mode에 대해서는 값을 set하고 함수를 종료한다.

코드 8-1. void setDP, setAP(int pin, int mode)

```
void setDP(int pin, int mode){
if(mode){
    switch(pin){
        case 0: //PC7
        *(volatile unsigned int*)(0x40023830) |= 0x01U<<2;
            //RCC Clock enable
        *(volatile unsigned int*)(0x40020800) |= 0x01U<<(7*2);
            //C_MODER - general purpose output
        *(volatile unsigned int*)(0x40020808) |= 0x03U<<(7*2);
            //C_OSPEEDR //11 - very high speed
        *(volatile unsigned int*)(0x4002080C) |= 0x01U<<(7*2);
            //C_PUPDR - pull up
        *(volatile unsigned int*)(0x40020818)  = 0x01U<<(7);
            //bit set
        break;
        case 1: //PC6
        //...
    }
}else{
    switch(pin){
        case 0: //PC7
        *(volatile unsigned int*)(0x40023830) |= 0x01U<<2;
        *(volatile unsigned int*)(0x40020800) &= ~(0x03U<<(7*2));
        *(volatile unsigned int*)(0x40020808) &= ~(0x03U<<(7*2));
        *(volatile unsigned int*)(0x4002080C) &= ~(0x03U<<(7*2));
        break;
        case 1: //PC6
        //...
    }
}
}
```

8.2 void outDP, outAP(int pin, int mode)

out*P 함수는 set*P함수를 사용하여 setting된 pin이 output mode일 때 set하거나 reset하는 함수이다. 내부는 pin에 대해 v가 1이면 set, 0이면 reset하도록 설계되어 있다.

코드 8-2. void outDP, outAP(int pin, int v)

```
void outDP(int pin, int v){
if(v){
    switch(pin){
        case 0: //PC7
        *(volatile unsigned int*)(0x40020818)  = 0x01U<<(7);
        break;
        case 1:
        //...
    }
}else{
    switch(pin){
        case 0: //PC7
        *(volatile unsigned int*)(0x40020818)  = 0x01U<<(7+16);
        break;
        case 1:
        //...
    }
}
```

8.3 int inDP, inAP(int pin)

in*P함수는 set*P함수를 사용하여 setting된 pin이 input mode일 때 해당 pin의 값을 읽어오는 함수이다. IDR 값을 shift하여 0번 비트로 옮긴 후 반환한다.

코드 8-3. int inDP, inAP(int pin)

```
int inDP(int pin){
int returnV = 0;
switch(pin){
    case 0:
    returnV = (*(volatile unsigned int*)(0x40020810U) >>7)&(0x01U);
    //PC7
    break;
    case 1:
    //...
}
return returnV;
}

```

8.4 unsigned int int_sqrt(unsigned long long n)

함수 이름에 명시된 대로 Square Root 값을 연산하는 함수이다. C언어에서는 math.h를 통해 연산을 제공하지만 math.h의 sqrt는 double에 대한 입출력으로 정의되어 있으나 본 프로젝트에서는 정수에 대한 Square Root를 연산하여야 한다. 또한 정확한 Square Root값을 요구하는 것이 아니기 때문에 유사치를 갖는 방법을 탐색하였으며, Newton-Raphson 방식의 루트 근사치 연산 방식을 찾을 수 있었다. Newton-Raphson을 통한 루트의 근사치는 아래의 수식에서 y가 x보다 커지면

해당 x를 루트의 근사치로 판명한다.

$$y_{n+1} = \frac{1}{2}\left(y_n + \frac{x}{y_n}\right)$$

별도의 C code를 통해 32,000,000는 15번의 반복 후에, 64,000,000,000,000은 18번의 반복 후에 return하는 것을 확인하였으며, 연산하는 값을 고려하였을 때 13~15번의 반복을 할 것으로 추정하였으며, 이를 통해 while문을 사용하여도 무방할 것으로 판단하였다.

입력 인수인 n의 경우 4byte인 int인 값의 제곱을 입력으로 받기 때문에 오버플로우가 발생하지 않도록 long long type을 통해 입력받았다. 연산량 최적화를 위해 나누기 2 대신 shift 연산을 사용하였다.

코드 8-4. unsigned int int_sqrt(unsigned long long n)

```
unsigned int int_sqrt(unsigned long long n) {
    unsigned long long x = n;
    unsigned long long y = (x + 1)>>1;

    while (y < x) {
        x = y;
        y = (x + n / x)>>1;
    }

    return (unsigned int)x;
}
```

8.5. void MyDelay_100ms(unsigned int n)

MyDelay_100ms 함수는 ~~강의 초반에 사용한~~ MyDelay의 변형으로 for loop에서의 최대값이 n*3,600,000으로 대략 n*0.1초 즉 n*100ms만큼 동작을 정지하는 효과를 갖는다. 이 함수는 구슬이 출발하기 전에 세팅이 끝난 후 모드 전환 후 잠시 대기 등의 역할을 위해 사용된다.

코드 8-5. void MyDelay_100ms(unsigned int n)

```
void MyDelay_100ms(unsigned int n){
    volatile unsigned int delay;
    for(delay=0;delay<=n * 3600000;delay++);
}
```

Ⅸ. 물리 모델

9.1. 첫 번째 구슬의 이동 시간

4.1.에서 첫 번째 구슬이 출발하는 지점부터 Sensor1, Sensor2, Actuator1, Actuator2의 거리를 S, $4S$, $9S$, $16S$로 정하였으며, 초기 속도가 0이므로 각 지점을 지나는 시간은 t_0로 일정함을 보였다.

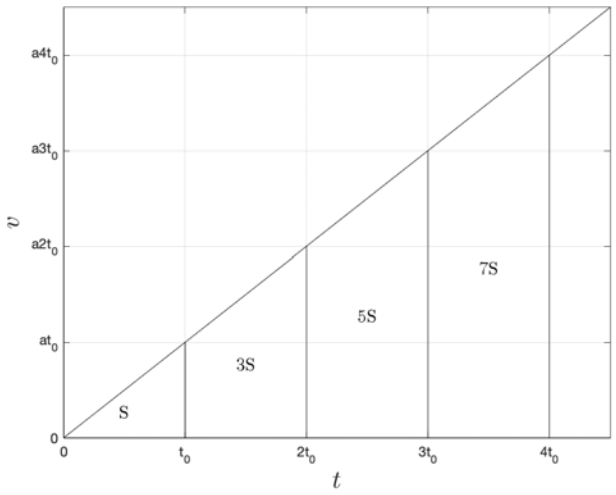


그림 9-1-1. 첫 번째 구슬의 시간에 따른 속도 그래프

이를 통해 Sensor1에 첫 번째 구슬이 감지되는 시점부터 Sensor2에 감지되는 시점을 통해 두 Sensor를 지나는 시간은 t_0에 해당하며, Sensor2에서 Actuator1까지 가는데 걸리는 시간과, Actuator1에서 Actuator2까지 가는데 걸리는 시간은 모두 t_0로 일정함을 알 수 있다.

9.2. d만큼 떨어진 구슬의 이동 시간

9.1.을 활용하여 첫 번째 구슬로부터 d만큼 떨어진 구슬이 각 지점을 지나는 시간에 대해 알아보자. 먼저 이를 위해 Sensor1, 2, Actuator1, 2를 지나는 시간을 t_{d1}, t_{d2}, t_{d3}, t_{d4}이라고 하자. t_{d1}는 $d+S$에 해당하는 시간일 것이고, t_{d2}는 $d+4S$, t_{d3}는 $d+9S$, t_{d4}는 $d+16S$에 해당하는 시간일 것이다. 이때 $v=at$이므로 $S=\frac{1}{2}at_0^2$이다. 또한 d에 대해서 원점까지 도달하는 시간을 t_d라고 하면 $d=\frac{1}{2}at_d^2$이 성립한다. d만큼 떨어진 구슬이 Sensor1에 해당하는 첫 번째 지점까지 이동하는 거리는 $d+S$이며, $2S=at_0^2$, $2d=at_d^2$라고 하면

$$at_{d1}^2=2d+2S=at_d^2+at_0^2$$

과 같이 나타낼 수 있다. 이를 통해 t_{d1}을 구하면

$$t_{d1}=\frac{\sqrt{at_d^2+at_0^2}}{\sqrt{a}}$$

이 성립한다. 이와 동일한 방식으로

$$t_{d2}=\frac{\sqrt{at_{d1}^2+3at_0^2}}{\sqrt{a}}$$
$$t_{d3}=\frac{\sqrt{at_{d2}^2+5at_0^2}}{\sqrt{a}}$$
$$t_{d4}=\frac{\sqrt{at_{d3}^2+7at_0^2}}{\sqrt{a}}$$

과 같이 나타낼 수 있다. 이때 $a=xm/s^2=1$이라고 하면

$$t_{d1}=\sqrt{t_d^2+t_0^2}$$
$$t_{d2}=\sqrt{t_{d1}^2+3t_0^2}$$
$$t_{d3}=\sqrt{t_{d2}^2+5t_0^2}$$
$$t_{d4}=\sqrt{t_{d3}^2+7t_0^2}$$

과 같이 나타낼 수 있다.

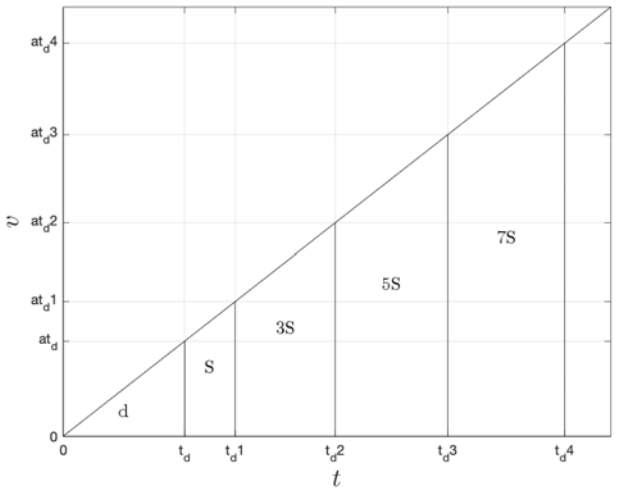


그림 9-1-2. d만큼 떨어진 구슬의 시간에 따른 속도 그래프

X. Logic 설계

10.1. 센서를 통한 시간 측정

첫 번째 구슬이 Sensor1에서부터 Sensor2에 도달하는 시간은 t_0로 나타낼 수 있다. 이때 각 지점을 지나는 시간은 모두 t_0로 같음을 보였다.

첫 번째 구슬이 Sensor1에 감지된 시점에서부터 d만큼 떨어진 구슬이 Sensor1이 감지된 시점까지의 시간을 구하면 이 둘을 더해 t_{d1}를 구할 수 있다. 동일한 방식으로 첫 번째 구슬이 Sensor1에 감지된 시점부터 d만큼 떨어진 구슬이 Sensor2에 감지된 시점까지의 시간을 구해 t_0와 더하면 t_{d2}를 구할 수 있다.

t_0와 t_{d2}를 구하였으므로 9.2.를 활용하면 t_{d3}와 t_{d4}를 구할 수 있다. 이때 t_{d2}를 센서를 통해 구할 수 있으므로 t_{d1}은 구할 필요가 없다.

10.2. 구슬간의 간격

내용 제시에서는 X간격이라 했으나 IX.를 통해 두 번째, 세 번째 구슬이 각각 X간격으로 떨어진 것이 아닌 n번째 구슬이 첫 번째 구슬과 d_n만큼 떨어져 있다고 정의하고 각 구슬의 첫 번째 구슬과의 간격을 d라고 판단하고 연산하도록 할 것이다. 이를 통해 간격이 X로 일정하게 떨어져 있는 것이 아닌 랜덤값의 간격을 가져도 동작할 수 있게 된다.

10.3. 구슬 이동성의 특징

10.1.을 사용할 경우 제일 먼저 t_0를 구할 수 있다. 첫 번째 구슬이 Sensor2를 지난 이후에 다음 구슬이 Sensor2를 지날 수 있으므로 t_{d2}는 이 다음에 구할 수 있다. 또한 t_{d2}를 구한 후에는 t_{d3}와 t_{d4}를 구할 수 있다. 이때 여러개의 구슬이 순차적으로 내려오므로 연속적으로 구해지는 각각의 d_n에 대한 t_{d3}와 t_{d4}는 정렬되어 있어야 하며, 필연적으로 정렬되어 있을 것이다.

10.4. Capture/Compare의 사용

10.6.에서 기술할 Timer의 사용에서 Timer는 overflow interrupt를 활용하는 것이 아닌 전체 시간을 CNT값을 통해 나타내도록 할 것인데, 이때 t_{d3}와 t_{d4}를 Capture/Compare를 통해 활용할 것이다. CC1과 CC2를 각각 t_{d3}와 t_{d4}를 타나내는데 사용할 것이며, 10.5.에서 기술하는 서보모터를 제어하는 타이밍에 활용한다.

10.5. 서보모터 제어

프로젝트 최초 설명에 따르면 n번째 구슬이 Actuator에 도달하는 시간에 맞춰 서보모터를 구동해야 하며, 서보모터의 구동에 시간이 걸리기 때문에 이를 고려해야한다. 이때 서보모터가 회전하는 시간을 t_{servo}라고 하고 Actuator1를 통과해야하는 상황이라고 하면 $t_{d3} - t_{servo}$일 때 서보모터의 구동을 시작해야 한다고 해석할 수 있다.

하지만 n-1번째 구슬이 해당하는 Actuator를 통과했다면 n번째 구슬이 Actuator를 통과하는 시간을 구해서 제어하는 것이 아닌 n-1번째 구슬이 해당 Actuator를 통과하였을 때 n번째 구슬이 해당 Actuator를 통과할 때 해야하는 동작을 수행해도 무방하다.

10.6. 각 interrupt의 작동 flow

본 프로젝트를 진행함에 있어 두 개의 Sensor에 의한 interrupt와 하나의 Timer에 의한 interrupt에 의해 동작하도록 설계하였는데, 10.1.~5.의 내용을 종합해 각 interrupt에서 수행해야 하는 logic에 대해 기술하도록 하겠다.

Timer는 첫 번째 구슬이 출발하는 시점을 0으로 하여 IX.과 10.1.~5.에서 기술한 시간을 나타내도록 한다.

Sensor1에 의해 첫 번째 interrupt가 발생한 경우는 첫 번째 구슬이 Sensor1을 지나는 시점이다. 이때 Timer를 활성화시켜 CNT값이 증가하도록 설정한다.

Sensor2에 의해 첫 번째 interrupt가 발생한 경우 또한 첫 번째 구슬이 Sensor2를 지나는 시점인데, 이때 CNT값을 통해 t_0를 구한다. 또한 현재 CNT는 첫 번째 구슬이 Sensor1을 지나는 시점이 0이므로 CNT값에 두 배를 취해 $2t_0$로 만들고, CNT가 0인 시점이 첫 번째 구슬이 출발지점에서 출발하는 시점으로 만든다.

또한 구슬이 여러개이기 때문에 9.2.에서 구한 t_{d3}와 t_{d4}를 지속적으로 구해야 하는데 이때 공통적으로 $5{t_0}^2$과 $7{t_0}^2$이 사용되므로 해당 값들에 대한 연

산도 진행한다. 또한 10.5.에 의해 n-1번째 구슬이 Actuator를 지나는 시간이 필요하므로 t_{d3}와 t_{d4}가 요구되는데, 이때 첫 번째 구슬의 t_{d3}와 t_{d4}는 $3t_0$와 $4t_0$이므로 Timer의 CCR1과 CCR2의 값을 $3t_0$와 $4t_0$의 값을 넣어 첫 번째 구슬이 각 Actuator에 도달하는 시점에 Timer가 interrupt를 발생시키도록 한다.

Sensor2에 발생하는 두 번째부터의 interrupt에 대해서는 Timer의 CNT값을 사용하여 n번째 구슬의 t_{d2}값을 구한다. t_{d2}값을 통해 t_{d3}와 t_{d4}를 구하고 이를 저장한다.

Timer의 경우 CCR1에 의한 interrupt와 CCR2에 의한 interrupt가 발생하는데 각 CCR은 t_{d3}와 t_{d4}를 나타내기 때문에 n번째 구슬이 Actuator1과 2에 도달했을 interrupt가 발생하게 된다. 이때, 다음 t_{d3}, 혹은 t_{d4}로 CCR을 갱신, 수정하여 다음 구슬이 도착하는 시점에 또다시 interrupt가 발생하도록 한다. 또한 n+1번째 구슬이 해당 Actuator에서 통과하는지 탈출하는지에 따라 서보모터를 동작시킨다.

XI. 주요 변수와 연산 함수

11.1. Indexing 규칙

주요 변수들 중에는 구슬의 순서에 따른 indexing이 사용되는 케이스가 있다. 이때 첫 번째 구슬의 index는 0, n번째 구슬의 index는 n-1로 사용한다.

11.2. 주요 변수와 역할

1. count[3] : 구슬이 출발하기 전 V.에서 기술한 인터페이스의 switch 입력을 통해 순서를 입력받는다. 이때 1~3번째 switch는 각각 1~3번 홀에 들어간다는 것을 의미하며, count는 각 홀에 들어가는 구슬의 숫자를 count한다.
2. count_rt[3] : 구슬이 출발하여 각 홀에 들어갈 때 마다 각 홀에 들어간 숫자를 나타내기 위한 배열이다.
3. order[20] : n번째 구슬이 들어갈 홀의 숫자를 저장한다. order[] 내의 값은 0~3이며, 1~3은 해당 홀에 들어가야함을 의미한다. count[]의 switch 입력에 대해 동작한다. order[5]가 1이라면 여섯 번째 구슬은 첫 번째 홀에 들어가야 한다. 구슬은 최대 20개까지 라고 가정하고 크기를 20으로 하였다.
4. order_num : order[]에 입력할 때 인덱스로 사용된다. switch 입력에 대해 order[order_num]에 해당하는 홀의 숫자를 저장한다.
5. sensor(n)_count, actuator(n)_count : 각 Sensor와 Actuator를 지나간 구슬의 숫자를 나타낸다. interrupt를 사용하여 10.3.에서 언급한 연속된 t_{d3}와 t_{d4}를 저장, 작성하는데 사용된다.
6. t0, t0t0, t0p5, t0p7 : 각각 t_0와 $5t_0^2$, $7t_0^2$의 값을 저장한다. Sensor2에 첫 번째 구슬이 감지되면 값들이 지정된다. 연산 방식과 과정은 IX.과 10.6.에서 기술하였다. 제곱으로의 연산, 제곱인 값을 저장하므로, int type으로는 overflow가 발생하여 오류가 발생할 가능성이 높으므로 long long type을 사용하였다.
7. td2[20], td3[20], td4[20] : 각각 n번째 구슬의 t_{d2}와 t_{d3}, t_{d4}를 저장한다. 연산 방식과 저장 및 사용 순서는 10.6.에서 기술하였다.

11.3. 연산 함수

9.2.에서 언급한 t_{d3}와 t_{d4}를 구하는 수식을 바탕으로 10.6.의 logic 내에서 t_{d3}와 t_{d4}가 연산되며, 이는 11.2.의 5, 6을 사용하여 7에 저장된다. 9.2.의 수식을 함수로 나타내면 다음과 같다

코드 11-3. unsigned int calculate_td(x)(unsigned int td(x-1))

```
unsigned int calculate_td3(unsigned int td2){
    unsigned int returnV;
    returnV = int_sqrt(((long long)td2*(long long)td2 + t0p5));
    return returnV;
}

unsigned int calculate_td4(unsigned int td3){
    unsigned int returnV;
    returnV = int_sqrt(((long long)td3*(long long)td3 + t0p7));
    return returnV;
}
```

위 두 함수는 각각 t_{d2}와 t_{d3}를 입력으로 받아 t_{d3}와 t_{d4}를 반환한다. 이때 8.4.에서 기술한 unsigned int int_sqrt()를 사용하는데, 11.2.의 6에서 언급한 overflow가 발생할 가능성이 높기 때문에 long long type으로의 casting을 하였으며, 실제로는 해당 함수들이 사용되지 않고 해당 연산이 이루어지는 Interrupt service routine인 void EXTI4_IRQHandler() 내에 inline으로 작성되어 있다.

XII. Procedural Function

12.1. int main()

대부분의 logic이 interrupt Handler 내에서 처리되기 때문에 main은 대부분이 초기화에 초점을 두고 있다. 서보모터 제어를 위한 TIM3와 이의 Ch1, Ch2, 해당 핀에 대한 setting과 Sensor1과 Sensor2의 interrupt 사용을 위한 setting, 32bit Timer를 사용하기 위한 TIM2의 세팅 함수가 포함된다.

코드 12-1. int main()

```
int main(void){
    hwInit();

    GPIO_Config_TIM3_PC6_ARD1_PWM();
    TIM3_PC6_ARD1_PWM_Config();
    GPIO_Config_TIM3_PC7_ARD0_PWM();
    TIM3_PC7_ARD0_PWM_Config();

    interrupt_setting_PG6_ARD2();
    interrupt_setting_PB4_ARD3();

    TIM2_setting();

    for(int i=4;i<11;i++)setDP(i,OUT);
    for(int i=0;i<4;i++)setAP(i,IN);

    setDP(11,OUT);
    outDP(11,led);

    check_servo();

    input_setting();

    setting_first_position();

    MyDelay_100ms(10);

    volatile unsigned int t;

    while(1){
        outDP(8,1);
        for(int k=0;k<4;k++)outDP(4+k,(count_rt[2]>>k) & 0x01);
        for(t=0;t<500;t++);
        outDP(8,0);

        outDP(9,1);
        for(int k=0;k<4;k++)outDP(4+k,(count_rt[1]>>k) & 0x01);
        for(t=0;t<500;t++);
        outDP(9,0);

        outDP(10,1);
        for(int k=0;k<4;k++)outDP(4+k,(count_rt[0]>>k) & 0x01);
        for(t=0;t<500;t++);
        outDP(10,0);
    }


    return 0;
}
```

또한 표 5-3-1의 핀의 입출력 설정 함수를 호출하며, void check_servo()를 통한 TIM3 세팅 완료 여부 및 와이어링 등 작동 상태 확인 과정을 거친다. void input_setting()에서는 switch 입력을 통해 n번째 구슬이 몇 번째 홀에 들어가는지 지정받는다.

10.5.에서 n번째 구슬이 지나갈 때의 서보모터 제어는 n-1번째 구슬이 지나갈 때 하도록 하는데, 이때 index가 0에 해당하는 첫 번째 구슬은 n-1이 존재하지 않는다. 때문에 첫 번째 구슬은 Timer Interrupt가 아닌, void input_setting() 함수가 종료된 후, void setting_first_position()을 통해 경사를 지나갈 수 있다.

위의 과정이 종료된 후, 약 1초간 멈췄다가 7-Segment가 착시 효과를 통해 각 홀에 들어간 구슬의 수를 display한다. 각 Segment 전원 pin을 set하고 7447IC에 값을 입력한 후 잠깐의 delay 후에 Segment의 전원을 끈다. 실질적으로 while loop에 들어간 이후에 구슬이 출발하도록 해야 한다.

12.2. void check_servo()

void check_servo()는 단순히 서보모터에 대한 setting이 잘 이루어졌는지 확인하는 함수이다. CCR1과 CCR2값을 바꿔 서보모터가 정상적으로 확인한다.

코드 12-2. void check_servo()

```
void check_servo(){
    *(volatile unsigned int*)(0x40000434) = 1000;
    MyDelay_100ms(2);
    *(volatile unsigned int*)(0x40000438) = 1000;
    MyDelay_100ms(2);
    *(volatile unsigned int*)(0x40000434) = 2000;
    MyDelay_100ms(2);
    *(volatile unsigned int*)(0x40000438) = 2000;
}
```

12.3. void input_setting()

switch를 통해 n번째 구슬이 들어가야 할 홀의 숫자를 입력받기 위한 함수이다. switch 입력은 polling으로 구동되며, now(n)에 while loop으로 입력을 받는다. loop의 끝에서 prev(n)에 now(n)의 값을 저장하여 두 값을 비교하여 switch 입력 여부를 판별하며 rising edge를 검출한다. 5.1.에서 기술하였다시피 1~3번 switch가 눌린 순서에 따라 order[]에 홀의 번호가 저장된다. 4번 switch가 눌리면 loop을 break하고 함수를 종료한다.

switch 입력을 받는 동안에는 12.1.의 int main()에서 while loop 내에서 7-Segment를 사용하여 통과된 구슬의 수를 display 했던 것과 동일하게 현재 각 홀마다 들어가야 할 구슬의 수를 display 한다.

코드 12-3. void input_setting()

```
void input_setting(){
    int now1 = 0;
    int now2 = 0;
    int now3 = 0;
    int now4 = 0;
    int prev1 = 0;
    int prev2 = 0;
    int prev3 = 0;
    int prev4 = 0;
    volatile unsigned int t;
    while(1){
        now1 = inAP(0);
        now2 = inAP(1);
        now3 = inAP(2);
        now4 = inAP(3);

        if(now1 && !prev1){
            *(volatile unsigned int*)(0x40000434) = 1000;
            count[0]++;
            order[order_num] = 1;
            order_num++;
        }
        if(now2 && !prev2){
            *(volatile unsigned int*)(0x40000434) = 2000;
            *(volatile unsigned int*)(0x40000438) = 1000;
            count[1]++;
            order[order_num] = 2;
            order_num++;
        }
        if(now3 && !prev3){
            *(volatile unsigned int*)(0x40000434) = 2000;
            *(volatile unsigned int*)(0x40000438) = 2000;
            count[2]++;
            order[order_num] = 3;
            order_num++;
        }
        if(now4 && !prev4)break;
        prev1 = now1;
        prev2 = now2;
        prev3 = now3;
        prev4 = now4;

        outDP(8,1);
        for(int k=0;k<4;k++)outDP(4+k,(count[2]>>k) & 0x01);
        for(t=0;t<500;t++);
        outDP(8,0);
        outDP(9,1);
        for(int k=0;k<4;k++)outDP(4+k,(count[1]>>k) & 0x01);
        for(t=0;t<500;t++);
        outDP(9,0);
        outDP(10,1);
        for(int k=0;k<4;k++)outDP(4+k,(count[0]>>k) & 0x01);
        for(t=0;t<500;t++);
        outDP(10,0);
    }
}
```

12.4. void setting_first_position()

10.5.의 n번째 구슬의 통과여부 판별 로직은 n-1번째 구슬이 해당 Actuator를 통과하는 시점에 이루어지는데, 이때 첫 번째 구슬은 n-1이 없기 때문에 void input_setting()함수가 끝나면 시스템은 바로 첫 번째 구슬이 통과하는 포지션을 가진다. 이때 CCR이 2000인 경우가 통과, 1000인 경우가 홀로 빠지는 경우이다.

만약 첫 번째 구슬이 첫 번째 홀에 들어가는 경우 다음 두 번째 혹은 세 번째 홀에 들어가는 경우에 따라 Actuator2를 미리 제어한다.

코드 12-4. void setting_first_position()

```
void setting_first_position(){
    if(order[0]==1){
        *(volatile unsigned int*)(0x40000434) = 1000;

        int order_index = 1;
        while(order[order_index] == 1)order_index++;
        if(order[order_index] == 0)return;
        else if(order[order_index] == 2)
            *(volatile unsigned int*)(0x40000438) = 1000;
        else if(order[order_index] == 3)
            *(volatile unsigned int*)(0x40000438) = 2000;

    }else if(order[0]==2){
        *(volatile unsigned int*)(0x40000434) = 2000;
        *(volatile unsigned int*)(0x40000438) = 1000;
    }else if(order[0]==3){
        *(volatile unsigned int*)(0x40000434) = 2000;
        *(volatile unsigned int*)(0x40000438) = 2000;
    }
}
```

X III. IQRHandler and setting

13.1. void TIM2_setting

void TIM2_setting()은 Timer2를 사용하기 위해 setting하는 함수이다. 10.6.에서 Timer2는 overflow interrupt를 활용하는 것이 아닌 CNT값 자체를 시간으로 활용할 것이라고 기술하였다.

Capture/Compare를 활용하기 위해 CCnIE를 enable 하였으며, Flag bit인 CCnIF를 clear하였다. CNT는 reset value가 0이지만 reset하였으며, Auto Reload Register의 reset value가 최대값이므로 별도로 설정하지 않았다.

Prescaler는 43200 - 1로 설정하였는데, 해당 block의 clock이 108MHz이므로 1/2500sec마다 CNT값이 1씩 증가한다. Prescaler value는 10800의 배수 중 4배인 값을 사용하였다.

Interrupt and Exception table에서 TIM2의 global interrupt의 value는 28임을 확인하였다.

CEN은 EXTI6에서 설정, set 하지 않았다.

코드 13-1. void TIM2_setting()

```
void TIM2_setting(){
    *(volatile unsigned int*)(0x40023840) |= 0x01U<<0;
        //RCC_APB1ENR-TIM2EN
    *(volatile unsigned int*)(0x4000000C) |= 0x01U<<1;
        //TIM2_DIER-CC1IE
        //output compare interrupt1
    *(volatile unsigned int*)(0x4000000C) |= 0x01U<<2;
        //TIM2_DIER-CC2IE
        //output compare interrupt2
    *(volatile unsigned int*)(0x40000010) &= ~(0x01U<<1);
        //CC1IF clear
    *(volatile unsigned int*)(0x40000010) &= ~(0x01U<<2);
        //CC2IF clear

    *(volatile unsigned int*)(0x40000024) = 0;
        //TIM2_CNT reset
    *(volatile unsigned int*)(0x40000028) = 43200 - 1;
        //TIM2_PSC
    *(volatile unsigned int*)(0x4000000C) |= 0x01U<<0;
        //TIM2_DIER-UIE
    *(volatile unsigned int*)(0x40000010) &= ~(0x01U<<0);
        //TIM2_SR-UIF

    *(volatile unsigned int*)(0xE000E100) |= 0x01U<<28;
        //NVIC -> 28
}
```

13.2. void EXTI9_5_IRQHandler()

Sensor1은 EXTI6와 연결되어 있는데, 첫 번째로 호출될 때 TIM2를 시작하면 된다. 그 외에는 다른 동작을 하지 않는다. 때문에 PR을 확인 후 set을 통해 clear, sensor1_count value를 통해 첫 번째 호출임을 확인하고 TIM2의 CR1 registerdptj CEN bit를 set하여 TIM2를 활성화 한다.

코드 13-2. void EXTI9_5_IRQHandler()

```
void EXTI9_5_IRQHandler(){
    if(*(volatile unsigned int*)(0x40013C14) & (0x01U<<6)){
        //PG6
        *(volatile unsigned int*)(0x40013C14) |= 0x01U<<6;
            //EXTI_PR-PR6 clear
        if(sensor1_count == 0){
            *(volatile unsigned int*)(0x40000000) |= 0x01U<<0;
                //TIM2_CR1-CEN
                //timer 시작
            sensor1_count = 1;  //sensor1 detect first ball
            return;

        }
    }
}
```

13.3. void EXTI4_IRQHandler()

코드 13-3. void EXTI4_IRQHandler()

```
void EXTI4_IRQHandler(){
if(*(volatile unsigned int*)(0x40013C14) & (0x01U<<4)){
    //PB4
    *(volatile unsigned int*)(0x40013C14) |= 0x01U<<4;
        //interrupt clear
    if(sensor2_count != 0){
        td2[sensor2_count] = *(unsigned int*)(0x40000024);
                            //n번째 구슬이 Sensor2에 감지된 시간
        td3[sensor2_count] =
        int_sqrt(((unsigned long long)td2[sensor2_count]
        *(unsigned long long)td2[sensor2_count] + t0p5));
        td4[sensor2_count] =
        int_sqrt(((unsigned long long)td3[sensor2_count]
        *(unsigned long long)td3[sensor2_count] + t0p7));
        //td2를 베이스로 td3와 td4 계산, 배열에 저장
        if(*(volatile unsigned int*)(0x40000034) == 0){
        *(volatile unsigned int*)(0x40000034) = td3[sensor2_count];
        }//CCR이 비어있으면 TIM2에서 td3를 갱신하려했지만
            //값이 없어서 0이었던 경우
        if(*(volatile unsigned int*)(0x40000038) == 0){
        *(volatile unsigned int*)(0x40000038) = td4[sensor2_count];
        }

        sensor2_count = sensor2_count + 1 ;
    }
    if(sensor2_count == 0){
        //check counter value and save value
        //caculate t_0
        t0 = *(volatile unsigned int*)(0x40000024); //TIM2_CNT
        *(volatile unsigned int*)(0x40000024) = t0 * 2;
    //modify CNT value
    //now CNT's 0 is first ball's start time
        t0t0 = t0 * t0;
        t0p5 = t0t0 * 5;
        t0p7 = t0t0 * 9;
            //원래는 9이지만
            //3d프린터 구조물에 의한 마찰변화로 인한 조정
        td3[0] = t0*3;
        td4[0] = t0<<2;
        *(volatile unsigned int*)(0x40000034) = td3[0];
        *(volatile unsigned int*)(0x40000038) = td4[0];
            //첫 번째 구슬이 Actuator1,2에 도착하는 시간 CCR에 입력
        sensor2_count = 1;
    }
}
}
```

Sensor2는 첫 번째 구슬이 도착했을 때 t_0 측정, 현재 시간을 구슬이 출발하는 시점을 기준으로 설정, $5t_0{}^2$과 $7t_0{}^2$을 구하고 첫 번째 구슬이 각 Actuator에 도착하는 시간을 CCR에 작성한다.

두 번째 구슬부터는 t_{d3}와 t_{d4}를 구하기 위해 11.3.의 함수를 inline으로 작성하였으며, else if를 사용할 경우 오작동하여 if문 만으로 작성하였는데, sensor2_count의 값이 0인 상황에서 1이 된 후 양수 판별 if문으로 갈 경우 오작동하기 때문에 0인지 확인하는 if문이 양수 판별 if문 이후로 갔다.

이때 4.3.에서 언급한 3D프린터의 경사에 의한 가속도 변화와 이에 따른 주파시간 변화로 인해 t_{d4}를 구할 때에는 $7t_0{}^2$보다 작은 값을 사용한다.

13.4.에 의해 두 번째 부터의 구슬이 도착하는 시간이 핸들링되는데, 이때 다음 구슬의 t_{d3}나 t_{d4}가 연산되지 않은 시점에는 0의 값을 가지며, 0을 CCR값으로 가질 경우 오류가 난다. 이 때문에 Timer2에 의해 CCR값이 0이 된 경우에는 EXTI4에서 즉각적으로 t_{d3}나 t_{d4}를 반영한다.

13.4. void TIM2_IRQHandler()

TIM2는 CCR을 통해 n번째 구슬이 도착하는 시간을 가지고 있으며, n-1번째 구슬이 통과할 때 n번째 구슬의 통과 여부를 판별, 서보모터를 작동시키는 역할을 한다. 또한 n번째 구슬의 t_{d3}와 t_{d4}는 배열에 저장되어 있으므로 n+1번째 구슬의 통과여부를 위해 CCR을 sensor(n)_count를 사용하여 다음 구슬의 t_{d3}와 t_{d4}로 작성한다. 또한 통과한 구슬의 수를 Display하기 위해 count_rt값을 수정하는 과정을 거친다.

코드 13-4. void TIM2_IRQHandler()

```
void TIM2_IRQHandler(){
if(*(volatile unsigned int*)(0x40000010) & (0x01U<<1)){
    //Output Capture1
    *(volatile unsigned int*)(0x40000010) &= ~(0x01U<<1);
        //interrupt clear
    actuator1_count++;
    *(volatile unsigned int*)(0x40000034) = td3[actuator1_count];
        //CCR1 값 갱신 다음 구슬이 도착하는 시점
    if(order[actuator1_count] == 1)
        *(volatile unsigned int*)(0x40000434) = 1000;
    if(order[actuator1_count] == 2
    || order[actuator1_count] == 3)
        *(volatile unsigned int*)(0x40000434) = 2000;
        //구슬이 통과할거면 열고, 빠질거면 막음
    if(order[actuator1_count-1] == 1)count_rt[0]++;
}
if(*(volatile unsigned int*)(0x40000010) & (0x01U<<2)){
    //Output Capture2
    *(volatile unsigned int*)(0x40000010) &= ~(0x01U<<2);
        //interrupt clear
    actuator2_count++;
    *(volatile unsigned int*)(0x40000038)
        = td4[actuator2_count];
        //CCR2 값 갱신 다음 구슬이 도착하는 시점
    if(order[actuator2_count] == 2)
        *(volatile unsigned int*)(0x40000438) = 1000;
    if(order[actuator2_count] == 3)
        *(volatile unsigned int*)(0x40000438) = 2000;
        //구슬이 통과할거면 열고, 빠질거면 막음
    if(order[actuator2_count-1] == 2)count_rt[1]++;
    if(order[actuator2_count-1] == 3)count_rt[2]++;
}
}
```

XIV. 상세 보고서를 마치며

2024년 6월 2일

임베디드SW 기말 프로젝트 상세보고서

제출자 이 진 성

14.1. 요약

기본 틀은 금속 프레임에 아크릴을 통해 그리드를 만들어 구성하였다. 센서는 레이저와 CdS를 통해 디지털 센서를 구현하였고, 액추에이터는 기어를 통해 회전하는 장치를 통해 구슬이 빠져나갈 수 있도록 구성하였다. 센서와 액추에이터의 프레임은 3D프린터를 사용하였다. 홀에 들어가는 순서를 정하기 위한 버튼, 그리고 개수를 표시하기 위한 7-Segment를 사용한 인터페이스를 구성하였다.

센서와 액추에이터 등 각 지점의 거리는 동일한 시간동안 이동하는 거리를 기준으로 만들어졌으며, 이를 통해 일정 거리 뒤에서 동시에 출발하는 구슬이 해당 지점들에 도착하는 시간을 구하기 위한 물리 모델을 설계하였다.

센서간의 시간차를 통해 첫 번째 구슬이 도달하는 시간을 구하고, 구슬간에 센서에 도달하는 시간을 통해 각 액추에이터에 도달하는 시간을 구하였다. 서보모터가 구동하기 시작하는 시간은 해당 구슬이 지점에 도착하는 시간에서 서보모터가 구동하는 시간을 빼서 구한 것이 아닌, 직전 구슬이 지나가는 시점에 동작을 시작한다.

타이머는 전체 시간보다 긴 시간을 주기로 하며, Capture/Compare 모드는 각 구슬이 액추에이터에 도달하는 시간을 활용하였으며, 배열에 각 구슬이 액추에이터에 도달하는 시간을 연속적으로 기록하고 사용하였다.

14.2. 개선점

구슬이 동시에 출발하게 하기 위해 아크릴을 사용하여 구조물을 설계하였으나, 완벽히 동시에 출발하지 않는다. 때문에 구슬이 출발하는 속도가 미세하게 달라지고, 이 때문에 동일한 시간에 출발해서 동일한 가속도를 갖는 경우 구슬간의 간격이 일정하게 유지되어야 하지만 간격이 달라지는 것을 볼 수 있다. 이 때문에 실적적인 문제 해결을 보이기 어려우며 오작동의 가능성이 높아진다. 때문에 서보모터를 사용해서 구슬의 출발을 막고 있는 구조물을 치우는 방식으로 설계했어야 했다고 판단한다.